VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2018 Martin Topolánek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Martin Topolánek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Shopsys s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

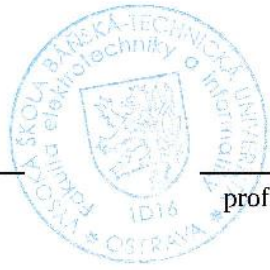
Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Lukáš**

Konzultant bakalářské práce: Ing. Tomáš Gottvald

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018



doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2018

Sopoláček

podpis

Souhlasím se zveřejněním této bakalářské/diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 30. dubna 2018

podpis

Chtěl bych poděkovat svému vedoucímu bakalářské práce Ing. Petru Lukášovi za jeho rady při vypracování této práce. Dále bych chtěl poděkovat svému konzultantovi bakalářské práce Ing. Tomáši Gottvaldovi za jeho vedení mé odborné praxe ve firmě Shopsys s. r. o. a také Janu Kročkovi, který byl mým mentorem při vývoji aplikace.

Abstrakt

Bakalářská práce shrnuje průběh odborné praxe ve firmě Shopsys s. r. o. Úvodem je popsán profil společnosti, ve které je praxe vedena. Následuje popis použitých technologií. Další část pojednává o specifikaci zadání aplikace. Na ni navazuje kapitola popisující zadané úkoly a jejich řešení. Závěr se věnuje znalostem, které jsem nabyl během studia a uplatnil v praxi. Poslední částí práce je celkové zhodnocení praxe.

Klíčová slova: odborná praxe, Shopsys, interní aplikace, informační systém, PHP, databáze

Abstract

This Bachelor thesis summarizes the course of professional practice at Shopsys s.r.o. company. At the beginning company profile is described. Description of used technologies is following. Next, specification of assigned application is described. Next capture is about required tasks and their solutions. Last but not least, there is a list of knowledges, which I gained during study and which were used during professional experience. At the end of this thesis whole experience evaluation is written.

Key Words: professional practice, Shopsys, internal application, information system, PHP, database

Obsah

Seznam použitých zkratek a symbolů

HTML	–	Hyper Text Markup Language
PHP	–	Hypertext Preprocesor
MySQL	–	My Structured Query Language
ER	–	Entity relationship

Seznam obrázků

Seznam výpisů zdrojového kódu

Úvod

Tato bakalářská praxe je vedena ve firmě Shopsys s. r. o. Firma expanduje do zahraničí a vznikl tak požadavek na interní aplikaci, která by zvládala evidenci a správu jejich zaměstnanců.

Obsah bakalářské práce bude zaměřený na tvorbu interní aplikace. Ve výsledku se pak bude jednat o samostatnou komponentu která bude spolu s ostatními komponentami tvořit jeden celek informačního systému. Náplní této práce pak bude tvorba aplikace, která by se stala takovou komponentou.

Celý sytém bude stavěn na jádru, který firma používa pro své internetové obchody.

1 O společnosti Shopsys s. r. o.

Firma existuje na českém trhu již 15 let. Vizí společnosti je profesionalita, progresivita a schopnost reagovat na moderní trendy vývoje elektronických obchodů. Mezi jejich hodnoty patří – důvěra, spolupráce, inovace a přátelskost. Jejich nabyté zkušenosti se snaží předávat dál prostřednictvím různým konferencí, jako je například PHP live. [1]

Významnými klienty jsou například OKAY s.r.o. a jejich dceřina společnost Jena - nábytek, s.r.o., B2B Partner s.r.o. a spousta dalších [2].

V poslední době se firma snaží také vyvinout vlastní knihovnu s otevřeným zdrojovým kódem pro elektronické obchodování. Tato knihovna pak ulehčí vývojářům jejich tvorbu internetových obchodů tak, aby se mohli plně soustředit na své zadání.

V této firmě jsem začínal na pozici kodéra. Náplní mé práce byl převod grafické šablony do HTML jazyka. Časem se má náplň práce rozšířila o programování v PHP a návrh databází.

2 Použité technologie

jQuery

Jedná o javascriptový framework, který si klade za cíl usnadit programátorům psaní často se opakujících příkazů. [3]

Smarty

Šablonovací systém navržený pro PHP programovací jazyk. [4] Umožňuje oddělit aplikační logiku od business logiky. Odstraňuje potřebu psát čistý PHP kód do HTML stránek.

```
{include file="menu.tpl"}
<div class="title">{$title}</div>
```

Výpis 2.1: Ukázka použití knihovny *Smarty*

Příklad výše (viz Výpis 2.1) demonstruje příklad použití knihovny *Smarty*. Vykreslí se obsah souboru `menu.tpl`. Následně se vypíše obsah proměnné `$title`.

Vývojové prostředí

Jako vývojové prostředí bylo použito NetBeans. Předností programu NetBeans byl našeptávač – pomocník při psaní kódu. Usnadňuje jeho psaní formou nabídky použití možných metod, atributů tříd, proměnných a podobně.

Práci zrychlovalo používání anotací. Ty zobrazují včetně nabídky také i nápovědu k metodám třídy.

Lokální server

Pro potřeby zprovoznění webové aplikace včetně databázového serveru bylo nutné použít lokální server. Server byl vytvořen za pomocí programu WAMP.

Verzovací systém

Pro účely zálohování byl použit jako verzovací systém Git. Správu verzování pak zajišťoval program TortoiseGit

3 Specifikace zadání

Vznikla potřeba na aplikaci pro správu uživatelských šablon pro vizitky a emailové patičky. Tyto šablony budou moci zpracovávat obrázky a pracovat s nimi. V aplikaci je dále zahrnuta následující funkčnost – správa firem, uživatelů a pracovních pozic.

Z důvodu expanze firmy je zároveň také požadavkem vícejazyčnost aplikace. Pro tyto potřeby je nutné zavést jazykové mutace stránek a jejich správu.

Aplikace je tvořena ve formě webového rozhraní za pomocí HTML, kaskádových stylů a skriptovacích jazyků – PHP a javascript. Možnost ukládání dat bude zajištěno pomoci databáze MySQL. Informační systém pro správu šablon bude spustitelný pouze na interní síti firmy Shopsys s. r. o.

Požadavkem na implementaci je, aby aplikace vycházela z jádra většiny internetových obchodů firmy Shopsys s. r. o. Tento požadavek vznikl na základě bohatých zkušeností ostatních programátorů s touto verzí systému.

3.1 Vstupy a výstupy

Vstupními údaji budou data o jednotlivých zaměstnancích, včetně informace pod hlavičkou, které firmy jsou vedeni. Uživatelé budou moci spravovat své vlastní údaje, údaje o firmách a šablonách.

Administrátor, případně superadministrátor, bude moci vkládat šablony jako HTML kód. Uživatel bude poté moci generovat HTML šablonu do PDF dokumentu, který bude výstupem aplikace.

3.2 Rozdělení webového rozhraní

3.2.1 Backend

Backend slouží pro správu, administraci webu a také pro zpracování dat. Zde budou mít uživatelé přístup, v závislosti na jejich uživatelských rolích (viz **3.3 Uživatelské role**), ke správě generovaných šablon, uživatelů a firem, které v aplikaci budou vedené.

3.2.2 Frontend

Bude dostupný pro všechny uživatele. Frontend bude sloužit pro generování šablon.

3.3 Uživatelské role

Přístup do aplikace budou mít 3 typy uživatelů:

3.3.1 Superadministrátor

Jedná se o takovou roli, která mění uživatelské přístupy a má k dispozici výpis všech uživatelů, včetně jejich aktuálního nastavení a taktéž přístup ke všem komponentám aplikace. Dokáže nastavovat role ostatním uživatelům. Nesmí však nastat situace, kdy by svou změnou role nezbyl v aplikaci žádný další superadministrátor. Jedná se o roli, která dokáže měnit jakékoliv nastavení, ať již uživatelů či aplikace.

3.3.2 Administrátor

Má přehled o historii generování vizitek a patiček do emailu. Dále má přístup k seznamu uživatelů vedených v aplikaci. Může upravovat nastavení aplikace, které přímo souvisí s generováním dokumentů – vkládání HTML šablon, editace názvu šablony a další funkce.

3.3.3 Uživatel

Bude mít přístup pouze na frontend k obsluze aplikace.

4 Seznam zadaných úkolů

4.1 Seznámení se s jádrem systému internetových obchodů

Prvním krokem k vytvoření interní aplikace pro zpracování šablon vizitek a emailových patiček uživatelů bylo zapotřebí proniknout do jádra informačního sytému většiny internetových obchodů firmy Shopsys s. r. o.

Této části jsem věnoval nejvíce času ze začátku praxe. Vzhledem ke komplexnosti a provázanosti systému bylo zapotřebí se k této časti v průběhu praxe vracet. Seznámení se s jáderm systému mi trvalo 10 dnů.

Body, týkající se této oblasti, o kterých se bude v této práci dále pojednávat:

- *autowiring*,
- třída `ServiceFactory`.

4.2 Návrh struktury databáze

Dalším krokem k vytvoření aplikace byl návrh datové vrstvy. Cílem bylo navrhnout takovou strukturu databáze, aby byla snadno rozšiřitelná o další komponentu do informačního systému. Tato struktura měla být zároveň navržena tak, aby splňovala kritéria vícejazyčnosti systému. Části s návrhem struktury jsem věnoval kolem 10 dnů.

Body, týkající se návrhu struktury databáze, o kterých se bude v této práci dále pojednávat:

- komponenty,
- šablony,
- vícejazyčnost.

4.3 Implementace aplikace

V poslední fázi bylo zapotřebí vytvořit aplikační a prezentační vrstvu aplikace. Implementace zahrnovala také napojení aplikační vrstvy na datovou. Implementaci aplikace jsem věnoval nejvíce svého času – odhadem 22 dnů.

Bod, kterému se bude práce dále věnovat:

- návrhový vzor *repository pattern*.

4.4 Kodérské úpravy

Jak již bylo uvedeno v úvodu, ze začátku odborné praxe bylo zapotřebí provést pár kodérských úprav. Toto období trvalo zhruba 8 dnů.

5 Řešení zadaných úkolů

5.1 Seznámení se s jádrem systému internetových obchodů

Jádro internetových obchodů firmy Shopsys s. r. o. prošlo za léta fungování vývojem, kdy jeho trend určovaly potřeby klientů. Důsledkem vzájemné interakce vznikl unikátní modulový systém.

Jedna z podmínek aplikace byl požadavek, aby jádro interní aplikace vycházelo právě z jádra těchto internetových obchodů a tím aby byla zajištěna srozumitelnost a udržitelnost kódu.

5.1.1 Autowiring

Jedním z mnoha prvků, které stojí za zmínku a na kterém jsou internetové obchody firmy Shopsys s. r. o. postaveny je tzv. *autowiring*.

Jedná se o mechanismus, který využívá reflexe. V případě tohoto mechanismu se jedná o čtení parametrů z konstruktorů a metod za účelem kontroly jejich datových typů. Pokud typ parametru je rozhraní a v aplikaci existuje více implementovaných rozhraní, nastává problém – není jasné, která implementace se má použít. Implementace služby *autowiring* vždy závisí na potřebách systému [5].

V případě tohoto jádra se *autowiring* kombinuje s třídou `ServiceFactory`, viz kapitola níže.

5.1.2 Třída ServiceFactory

Třída zajišťující vytvoření právě jedné instance třídy – buď to na požadavek služby *autowiring* nebo vytvoření přes metodu třídy `ServiceFactory`. Při každém dalším přístupu k instanci třídy se vrací již jednou vytvořená instance.

Vytvoření instance přes službu *autowiring* je vysvětleno níže, viz Výpis 5.1.

```
<?php
class CompanyService {
   private $database;
   public function __construct(Database $database) {
      $this->database = $database;
   }
}
?>
```

Výpis 5.1: Ukázka třídy CompanyService

Jedná se pouze o demonstrativní příklad, proto třída neobsahuje všechny metody a atributy.

Při volání konstruktoru třídy `CompanyService` služba *autowiring* pomocí reflexe zjistí třídu parametru `$database` a zavolá požadovanou `get` metodu v třídě `ServiceFactory`. V tomto případě by se jednalo o metodu `getDatabase`. V metodě se zkontroluje, zdali je instance třídy `Database` již vytvořená. Pokud ano – vrátí její instanci. V opačném případě se vytvoří nová instance třídy `Database` a uloží se do asociativního pole, respektive hashovací tabulky, kde klíčem je název třídy, v našem případě `Database`, a hodnotou je instance třídy. Poté metoda tuto instanci vrátí.

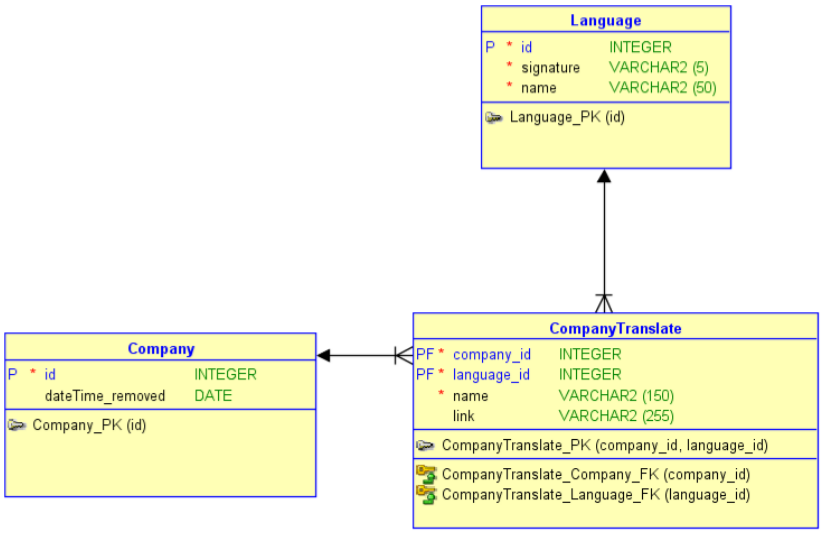
5.2 Návrh struktury databáze

5.2.1 Komponenty

Při návrhu databázové struktury pro komponenty bylo třeba dbát dvou požadavků – snadná rozšiřitelnost o další komponenty a jejich vícejazyčnost.

Tyto požadavky se odrážejí i na struktuře databáze. Ta pak obsahuje co nejméně závislostí mezi entitami.

Pro příklad je níže uveden ER diagram komponenty pro správu firem (viz Obrázek 5.1).



Obrázek 5.1: ER diagram komponenty pro správu firem

Na příkladu jsou uvedené tři entity – `Language`, `Company` a `CompanyTranslate` (uvedené entity obsahují pouze atributy určené pro demonstraci struktury databáze).

Language – entita, jenž je součástí jádra systému, tzn. společná entita pro všechny komponenty interní aplikace. Obsahuje atributy, které identifikují jazykovou entitu – identifikační číslo, zkrácený zápis a název jazyka.

Company a **CompanyTranslate** – jsou entity sloužící pro uchovávání dat o firmách a jejich překladech.

Atributy v entitě **Company** představují data společné pro všechny jazykové mutace, do čehož se řadí také vlastnost nazývající se `dateTime_removed`. Jedná se o příznak, který řeší funkci smazání záznamu z tabulky. Výchozí hodnota je nastavena na hodnotu `0`. V případě požadavku na samzání entity se tento atribut nastaví na aktuální datum a čas. Nedojde tak ke smazaní entity. V dotazích sloužících pro výběr záznamů z entit **Company** a **CompanyTranslate** se pak firma, ani její překlad, nezahrnuje do výsledku.

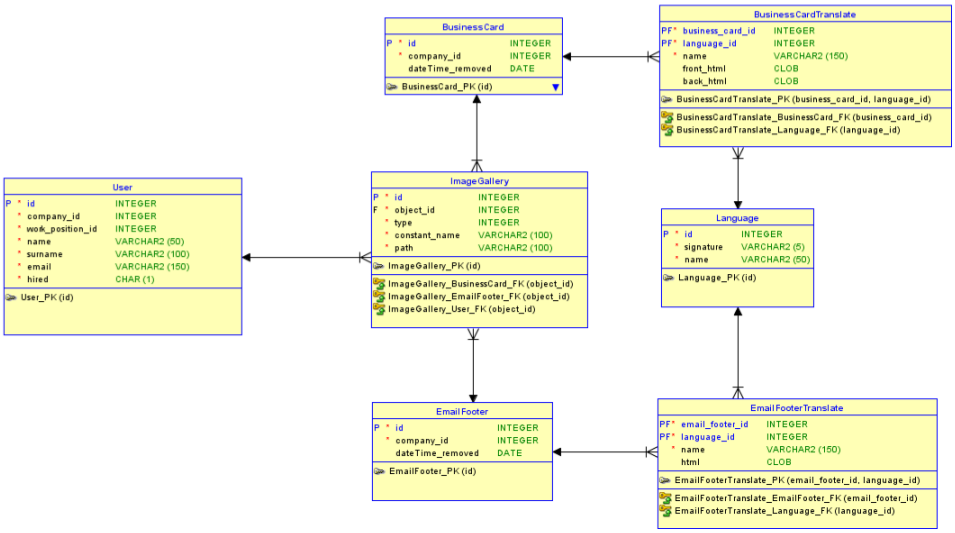
Entita **CompanyTranslate** pak obsahuje jazykové překlady firmy. V tomto případě se jedná o atributy `name` a `link`. K překladům se pak přistupuje přes identifikační číslo firmy a jazyka.

5.2.2 Šablony

Jak již bylo zmíněno v zadání specifikace – datová struktura měla být navržena kromě zpracování HTML šablon také i pro práci s obrázky pro šablony. Mezi takové entity, které pracují se šablonami i obrázky, patří:

- **BusinessCard** – uchovává data pro šablony vizitek,
- **EmailFooter** – uchovává data pro šablony patiček do emailů.

Entita **User** pak pracuje pouze s obrázky.
Následuje ER diagram zjednodušeného návrhu databázové struktury pro zpracování obrázků (viz Obrázek 5.2).



Obrázek 5.2: Zjednodušený návrh databázové struktury pro zpracování obrázků

Na obrázku lze vidět 7 různých entit s tím, že o významu entity **Language** bylo již zmíněno v předchozí kapitole.

BusinessCard a **BusinessCardTranslate** fungují jako vícejazyčná komponenta (viz předchozí kapitola **5.2.1 Komponenty**), která zpracovává šablony pro vizitky. Podobně fungují také entity **EmailFooter** a **EmailFooterTranslate**, s tím rozdílem, že tyto entity tvoří komponentu zpracovávající šablony pro patičky do emailu. Tyto komponenty ve srovnání s komponentou z předchozí kapitoly obsahují navíc atributy:

- `company_id` – identifikuje, k jaké firmě se daná šablona vztahuje,
- `front_html` – HTML šablona pro přední stranu vizitky,
- `back_html` – HTML šablona pro zadní stranu vizitky,
- `html` – HTML šablona pro patičky do emailu.

Entita **User** slouží jako základ pro komponentu spravující uživatele. Poslední nezmíněná entita z výše uvedeného příkladu – **ImageGallery** – obsahuje data obrázků šablon. Má následující atributy:

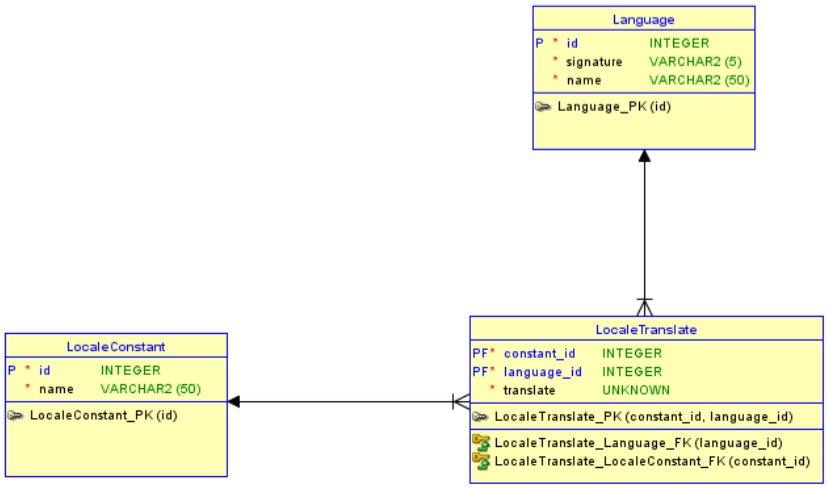
- `id` – identifikátor obrázku,
- `object_id` – identifikátor šablony nebo uživatele, ke kterému se obrázek vztahuje,
- `type` – identifikátor určující typ objektu (vizitka, patička do emailu, uživatel),
- `constant_name` – název textové konstanty, pod kterou bude dostupný v HTML šabloně,
- `path` – cesta k souboru.

5.2.3 Vícejazyčnost

Ve specifikaci je zmíněn požadavek na vícejazyčnost systému. Jedná se o entity, vztahující se k jádru systému. Pro účel internetových obchodů jsou vytvořeny dvě úložiště – lokální a centrální. Lokální slouží pro správu konstant danému internetovému obchodu. Centrální pak funguje jako globální úložiště, které obsahuje konstanty ze všech obchodů.

Jejich použití je triviální. Přístup k překladu takové jazykové konstanty je přes její název. Aplikace pak na základě názvu a aktuálně nastaveného jazyka zvolí vhodný překlad.

Níže se nachází zjednodušená struktura centrálního a lokálního úložiště (viz Obrázek 5.3).



Obrázek 5.3: Návrh datové struktury pro jazykové konstanty

Entita **LocaleConstant** obsahuje následující atributy:

- `id` – jednoznačný identifikátor konstanty,
- `name` – název konstanty, kterým se konstanta volá.

Entita **LocaleTranslate** obsahuje atributy:

- `constant_id` – jednoznačný identifikátor konstanty z entity `LocaleConstant`,
- `language_id` – jednoznačný identifikátor jazyka z entity `Language`,
- `translate` – překlad konstanty `constant_id` pro daný jazyk uvedený v `language_id`.

5.3 Implementace aplikace

5.3.1 Repository pattern

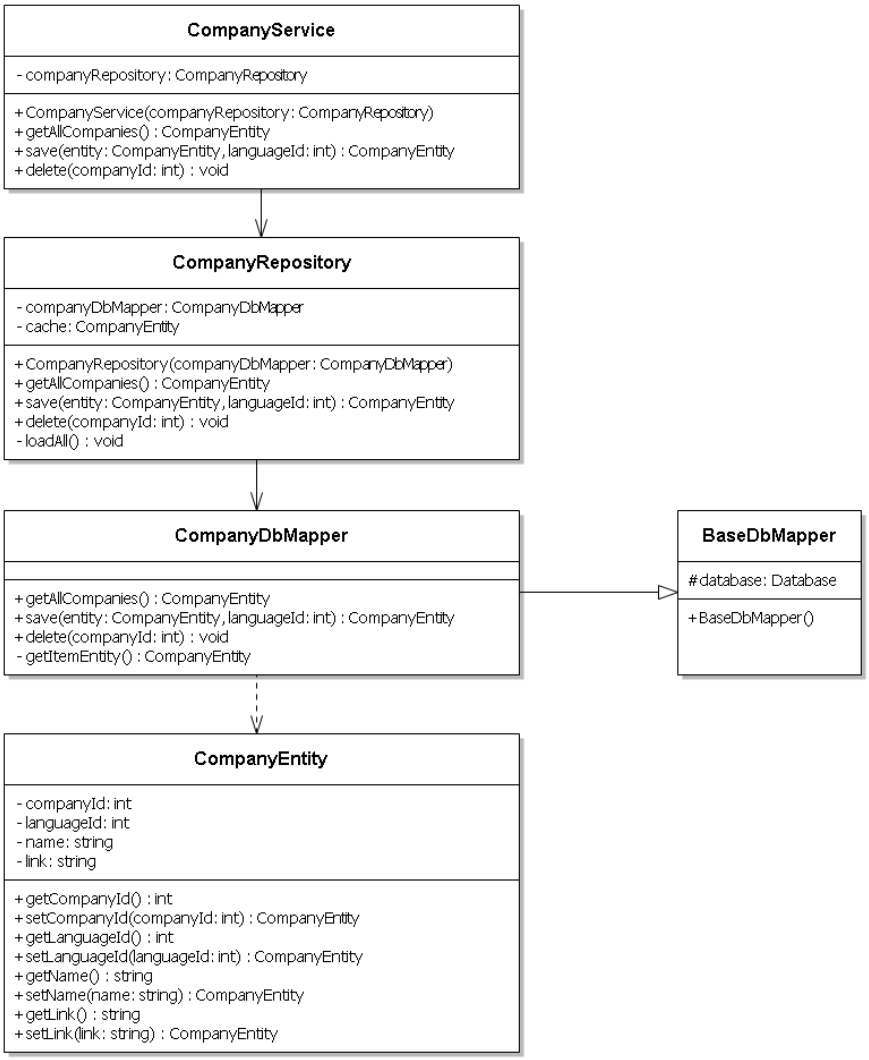
Repository pattern částečně řeší vytížení databáze dotazy pro výběr záznamů. Při prvním průchodu totiž načte data do repozitáře a při dalších dotazech je nenačítá z databáze, ale z lokálního úložiště v repozitáři.

Návrhový vzor se skládá ze tří základních prvků – entity, repozitáře a návrhového vzoru *data mapper*. **Entita** na aplikační vrstvě obsahuje tytéž atributy, co entita na datové vrstvě. Zároveň si udržuje jejich stavy. **Repozitář** slouží jako úložiště pro entity. Obsahuje také odkaz na *data mapper*, se kterým repozitář pracuje. ***Data Mapper*** se pak stará o dotazy na databázi z aplikační vrstvy. Ze získaných záznamů z databáze pak vytváří entity, které následně předává repozitáři.

5.3.2 Komponenta

Komponenta vychází z návrhového vzoru *repository pattern*. Entita na aplikační vrstvě je tvořena privátními atributy. Počet a názvy atributů entity na aplikační vrstvě se shodují s atributy entity na datové vrstvě. Obsahuje také metody buďto pro nastavení hodnoty atributu nebo vrácení hodnoty atributu – jedná se o metody `get` a `set`. V metodách `set` zároveň dochází k přetypování vstupního parametru na datový typ atributu, kterému se hodnota nastavuje. Komponenta dále rozšiřuje návrhový vzor o třídu *`NazevTridy`*`Service`.

Níže (viz Obrázek 5.4) se nachází třídní diagram komponenty pro správu firem. Jedná se pouze o demonstraci komponenty, proto třídy neobsahují všechny metody a atributy.



Obrázek 5.4: Třídní diagram komponenty pro správu firem

Služba `CompanyService` se stará o aplikační logiku nad entitami. Obsahuje mimo jiné privátní atribut `companyRepository`, který je referencí na instanci třídy `CopmanyRepository`. Ta je zajištěna pomocí již zmíněné služby `ServiceFactory` s kombinací se službou *autowiring*.

Repozitář `CompanyRepository`, jak už bylo zmíněno v kapitole **5.3.1 Repository pattern**, slouží pro ukládání entit do interního úložiště. Úložiště je řešeno formou hashovací tabulky, která se nachází v privátním atributu `$cache`. Klíčem této hashovací tabulky je zpravidla jednoznačný identifikátor entity a hodnotou je daná entita. Atribut v repozitáři `$companyDbMapper`, podobně jako atribut `$companyRepository` třídy `CompanyService,` obsahuje referenci. Přes konstruktor se do atributu `$companyDbMapper` ukládá reference na *data mapper*, třídy `CompanyDbMapper`.

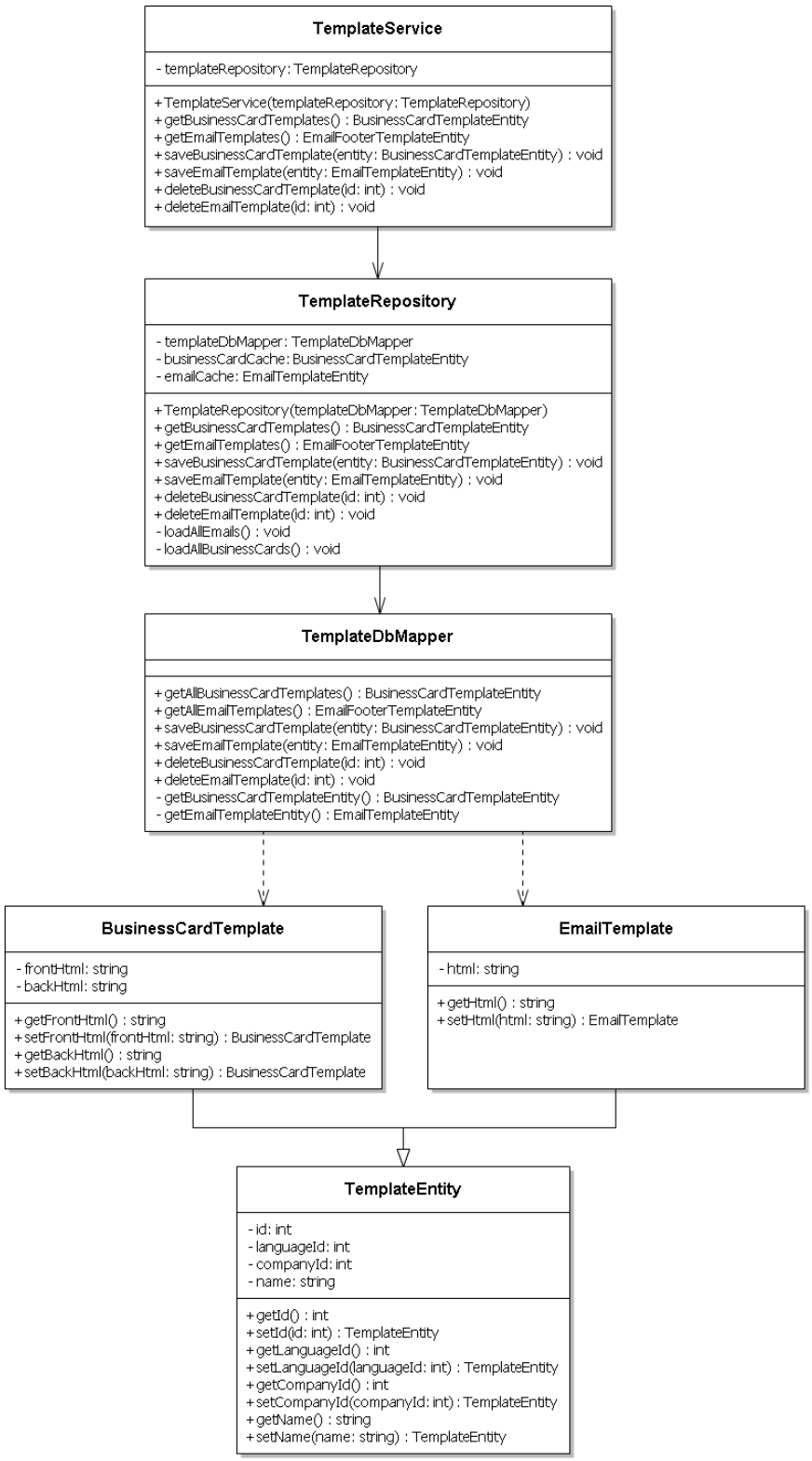
Data mapper je v demonstrujícím příkladě reprezentovaný třídou `CompanyDbMapper`. Tato třída je rozšířením třídy `BaseDbMapper`, ve které chráněný atribut `$database` má nastavené připojení k databázi. Připojení k databázi je řešeno v konstruktoru přes službu `ServiceFactory`.

`CompanyDbMapper` pak obsahuje metodu `getAllCompanies`, která vybírá záznamy firem z databáze a pomocí metody `getItemEntity` vytváří ze záznamů entity. Dále obsahuje metodu `save`, která entitu uloží a vrátí ji třídě, která metodu volala. V tomto případě třídě `CompanyRepository`. Metoda `delete` nastavuje v entitě `Company` na datové vrstvě atribut `dateTime_removed` na aktuální datum. Takový záznam pak nebude zahrnován ve výsledcích sloužící pro výběr záznamů z datové vrstvy.

`CompanyEntity` je následně třídou, která mapuje atributy entity z datové vrstvy za pomocí metod `get` a `set`, kterými se získávají a nastavují její atributy.

5.3.3 Šablony

Správa šablon je řešena komponentou `Template`. Komponenta se skládá ze služby `TemplateService`, repozitáře `TemplateRepository`, *data mapperu* `TemplateDbMapper` a dvou entit `BusinessCardTemplateEntity` a `EmailTemplateEntity`, které dědí ze společné třídy `TemplateEntity`. Níže se nachází třídní diagram této komponenty (viz Obrázek 5.5).



Obrázek 5.5: Třídní diagram komponenty `Template`

Komponenta `Template` se odlišuje od ostatních komponent pouze tím, že obsahuje více entit a také abstraktní třídu, ze které entity dědí. Abstraktní třída `TemplateEntity` tak obsahuje všechny společné atributy, které entity pro šablony vizitek a emailových patiček mají.

Nahrávání obrázků k šablonám a uživatelům má na starosti třída `ImageUpload`, která obsahuje následující veřejné metody:

- `initialize` – metoda volaná se dvěma vstupními parametry. První specifikuje složku kam nahrávat obrázky a druhý o jaký typ obrázku se jedná – foto uživatele, obrázky k šablonám vizitky nebo patičky do emailu. Uloží obrázky, které jsou určené ke smazání a které jsou určené k nahrání na server do interního úložiště.

- `process` – u každého obrázku zkontroluje jeho velikost, zdali je menší jak maximální povolená nahrávací velikost. Provede se kontrola formátu a kontrola zdali je vyplněný název konstanty. V případě, že je obrázek bezvadný, vytvoří se na základě získaných údajů o obrázku entita. Ta se pak uloží do interního úložiště. Jako poslední krok se zvaliduje název konstanty, zdali není duplicitní. V případě, že se v průběhu zpracovávání obrázku vyskytne chyba – vytvoří se varovná hláška zavoláním metody `setErrorMessage` třídy `MessageReporting`.

- `saveImages` – pro každý obrázek načtený z interního úložiště, který prošel procesem validace, se nastaví jednoznačný identifikátor, určující ke které šabloně, případně uživateli, se vztahuje. Dále se zavolá metoda `save` třídy `GalleryService` a do parametrů se vloží entita s obrázkem a příznak toho, k jakému typu objektu se vztahuje – zdali jedné ze šablon nebo uživateli.

- `deleteImages` – odstraní obrázky.

5.3.4 Vícejazyčnost

Webové rozhraní získává nové a spravuje stávající jazykové konstanty za pomocí třídy `ConstantManager` a o jejich vykreslování do HTML stránek se pak stará statická třída `Localize`. `ConstantManager` — je třída, která se používá zpravidla v administrační části, kde administrátor nebo superadministrátor může přidávat a upravovat jazykové konstanty. Metody třídy, které stojí za zmínění:

- `update` – bezparametrická metoda, která provede synchronizaci konstant mezi lokálním a centrálním úložištěm – nahraje nové konstanty v lokálním úložišti do centrálního a naopak. Poté přepíše překlady konstant, které v lokálním úložišti dosud nebyly upravené, překlady z centrálního úložiště. Poslední krok se děje pouze pro upravené konstanty v centrálním úložišti.

- `save` – bezparametrická metoda, která se stará o ukládání jazykové konstanty do databáze. Provedou se kontroly – zdali název konstanty vyhovuje požadovanému formátu a zdali existuje pro konstantu alespoň jeden překlad. V případě, že nebylo vyhověno jedné z podmínek, metoda se ukončí a vypíše chybovou hlášku. V opačném případě se konstanta uloží, případně přepíšou překlady, pokud existuje konstanta s již daným identifikačním názvem.
- `findWithParameters` – metoda má jako vstupní parametr asociativní pole, tedy hashovací tabulku. Tato tabulka obsahuje dvě hodnoty. Jeden z parametrů obsahuje hledaný název konstanty a druhý překlad konstanty. Na základě těchto hodnot metoda vrátí výsledný seznam jazykových konstant.

`Localize` — statická třída, která obsahuje privátní atribut `$data`. Tento atribut existuje jako hashovací tabulka, kde klíčem je název konstanty. Metoda `translate` pak dostává jako vstupní parametr název konstanty a vrací její překlad. Metoda `loadTranslatesData` v závislosti na aktuálně nastaveném jazyce načte jazykové konstanty do atributu `$data`.

Způsoby použití

Liší se v závislosti, o jakou část aplikace se jedná. V případě frontend části se použití kombinuje s knihovnou Smarty (viz 2 **Použité technologie**).

```
<h1>
   <#login|capitals#>
</h1>
```

Výpis 5.2: Ukázka použití jazykového překladu v kombinaci s knihovnou Smarty

Při vykreslování HTML stránky se pro každý výraz uzavřený ve značkách `<#` a `#>` aplikuje následující algoritmus (příklad takového výrazu je uvedený ve výpisu 5.2):

1. Řetězec se rozdělí na dva podřetězce. Oddělovacím znakem je znak '|'.
   První podřetězec se uloží do proměnné `$constant`, druhý pak do proměnné `$modifier`. V příkladě z výpisu 5.2 by pak proměnné `$constant` a `$modifier` obsahovaly řetězce `login` a `capitals`.
2. Zavolá se statická metoda `translate` třídy `Localize`, kde vstupním parametrem metody je proměnná `$constant`. Metoda vrátí přeložený řetězec, který se uloží do proměnné `$out`.

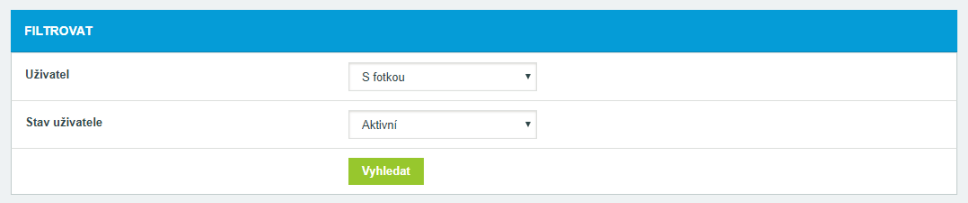
3. Na základě modifikátoru jazykové konstanty, uloženého v proměnné `$modifier`, se zavolá příslušná statická metoda ve třídě `SmartyCustom`. Vstupním parametrem této metody je proměnná `$out` a pokud to metoda vyžaduje, tak i další potřebné parametry. Na základě proměnné `$modifier`, v tomto případě, by se zavolala statická metoda, která by převedla první znak v řetězci na velké psímeno. Takový řetězec, který vrátí metoda, se uloží do proměnné `$out`.

4. Obsah proměnné `$out` se vloží do HTML stránky namísto výrazu, který se momentálně zpracovával.

Backend část aplikace nevyužívá k vykreslení HTML stránek knihovnu *Smarty*. Z toho důvodu se k překladu jazykové konstanty volá přímo statická metoda `translate` třídy `Localize`.

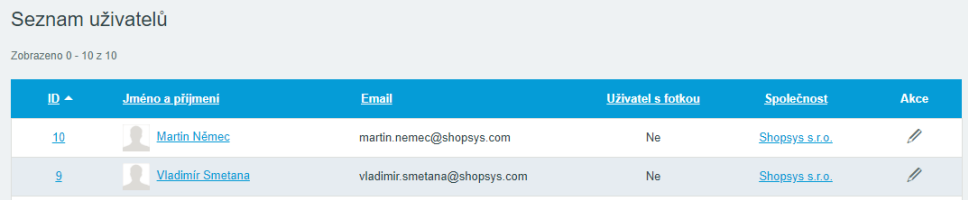
5.3.5 Uživatelé

Uživatelé jsou spravováni za pomocí komponenty `User`. Seznam zobrazených uživatelů je určen dvěma prvky, na základě zvoleného filtru (viz Obrázek 5.6) a zvolené stránky. V případě této aplikace se na stránce zobrazuje pouze 20 výsledků.



Obrázek 5.6: Filtrace uživatelů

Seznam jde dál seřazovat na základě jednotlivých sloupců a jejich hodnot v nich uvedených (viz Obrázek 5.7). Veškeré tyto služby – stránkování, filtrace a seřazování uživatelů – řeší třída `TableHeaderService` a k tomu příslušná entita `PagingEntity`.



Obrázek 5.7: Ukázka výpisu seznamu uživatelů

Stránkování i filtrace je řešeno pomocí třídy `TableHeaderService`. Obsahuje metody pro vytvoření stránkování, filtrace nebo hlavičky pro tabulku. Hlavička obsahuje sloupce, které se dají sestupně a vzestupně řadit dle hodnot v nich uvedených.

Entita `PagingEntity` obsahuje metody pro nastavení a získání čísla aktuálně prohlížené stránky, počtu záznámů na stránce, celkový počet stránek, seřazení záznamů a jejich filtrace. Služba `TableHeaderService` pak zpracovává data z entity a vykresluje filtr, hlavičku tabulky a stránkování.

5.4 Kodérské úpravy

Častým klientem, pro které byly prováděny kodérské úpravy během odborné praxe, byla společnost Okay s. r. o. Jedná se o velmi specifický projekt, který se odlišuje svou strukturou. Proto bylo zapotřebí dalšího nastudování systému, na kterém je stavěný.

6 Využití znalostí získaných během studia

Při vývoji webových aplikací ve firmě Shopsys s.r.o. se používá kromě skriptovacího programovacího jazyka PHP také skriptovací jazyk Javascript, se kterým jsem měl možnost se seznámit již v předmětu Vývoj internetových aplikací.

Při implementaci jsem využil také návrhové vzory, kterým jsem měl možnost porozumět již v předmětu Úvod do softwarového inženýrství. Tyto vědomosti urychlily pochopení jádra systému, na kterém jsem pracoval a zároveň zjednodušily psaní kódů, tak aby byly čitelné, srozumitelné a především výkonné.

Uplatnil jsem také znalosti z předmětu Vývoj informačních systémů, z něhož jsem čerpal při správné volbě vzorů, pro efektivní zpracování dat z databáze. Při tvorbě aplikace jsem také využil znalosti databázových systémů, k čemuž jsem využil nabytých znalostí z předmětu Úvod do databázových systémů.

7 Závěr

Možnost absolvovat odbornou praxi považuji za výhodu. Pomohlo mi to jednak zdokonalit mé technické znalosti v objektově orientovaném programování, návrhových vzorech, databázových systémech, ale především jsem získal široké znalosti a jednak schopnosti pracovat v týmu. Navíc v týmu lidí, kteří mě v oblasti programování a kodóvání webových stránek posunuli vpřed v mém odborném rozhledu.

Během mé odborné praxe jsem tak vytvořil solidní základ pro aplikaci, která umí spravovat firmy, uživatele a jejich pracovní pozice. Aplikace, která umožňuje správu šablon pro vizitky a patičky do emailu a jejich následný výstup pro uživatele ve formě PDF dokumentu.

Literatura

[1] O Shopsys. Shopsys [online]. ©2003-2018 [cit. 2018-04-30]. Dostupné z: https://www.shopsys.cz/o-shopsys

[2] Vybrané reference. Shopsys [online]. ©2003-2018 [cit. 2018-04-30]. Dostupné z: https://www.shopsys.cz/vybrane-reference

[3] ČÁPKA, David. 1. díl - Úvod do jQuery. ITnetwork.cz [online]. ©2018 [cit. 2018-04-16]. Dostupné z: https://www.itnetwork.cz/javascript/jquery-zaklady/javascript-tutorial-funkcionalni-programovani-a-jquery-webova-kalkulacka

[4] Chapter 1. What is Smarty?. Smarty [online]. 2018 [cit. 2018-04-17]. Dostupné z: https://www.smarty.net/docsv2/en/what.is.smarty.tpl

[5] PURCHART, Vašek. Dependency Injection: kontejner. Zdroják [online]. [cit. 2018-04-16]. Dostupné z: https://www.zdrojak.cz/clanky/dependency-injection-kontejner/